

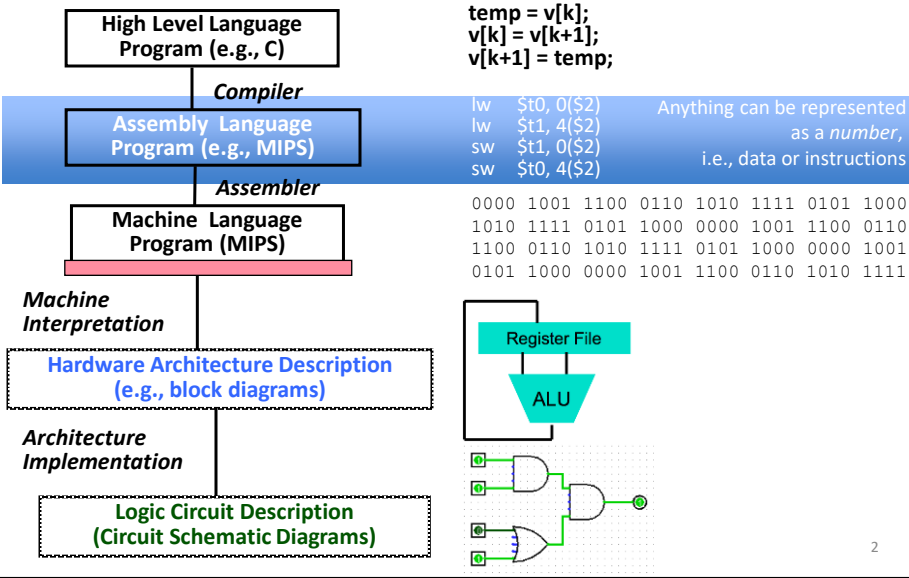
计算机组成

机器语言(1)

高小鹏

北京航空航天大学计算机学院

Levels of Representation/Interpretation



- 北京航空航天大学计算机学院

- 大体涉及5个环节：编译、汇编、存储、加载、运行

C程序	①编译	MIPS汇编程序	②汇编	指令的机器码 (2进制格式和16进制格式)
-----	-----	----------	-----	-----------------------



从C代码到可执行文件运行^{2/6}

□ CPU不能直接执行C源代码，必须利用编译器将C源代码转换为对应的汇编程序

- ◆ C源代码：其描述方式是适合人书写和阅读的
- ◆ 汇编代码：用更加接近CPU可以理解和执行的语言编写的程序

```

while ( 1 )
    if ( *p ) {
        *q = *p ;
        p++ ;
        q++ ;
    }
    else
        break ;

```

C程序

①编译

MIPS汇编程序

```

Loop_Start:
    lbu    $t1, 0($s1)
    sb     $t1, 0($s2)
    addiu  $s1, $s1, 1
    addiu  $s2, $s2, 1
    beq    $t1, $0, Loop_End
    j      Loop_Start
Loop_End:

```

编译相关内容属于
编译器技术范畴

5

从C代码到可执行文件运行^{3/6}

□ CPU也不能直接执行汇编代码，必须利用汇编器将汇编代码转换为对应的二进制机器码

- ◆ 汇编程序：每行对应一条机器指令
- ◆ 机器指令：由一组二进制01串组成，是CPU可以理解与执行的

```

Loop_Start:
    lbu    $t1, 0($s1)
    sb     $t1, 0($s2)
    addiu  $s1, $s1, 1
    addiu  $s2, $s2, 1
    beq    $t1, $0, Loop_End
    j      Loop_Start
Loop_End:

```

MIPS汇编程序

②汇编

指令的机器码 (2进制格式和16进制格式)

```

10010000001111100000000000000000 0x903E0000
10100010010010010000000000000000 0xA2490000
00100110001100010000000000000001 0x26310001
00100110010100100000000000000001 0x26520001
00010001001000000000000000000001 0x11200001
00001000000100000000000000000000 0x08100000

```

汇编相关内容属于
编译器技术范畴

6

从C代码到可执行文件运行^{4/6}

- C程序被编译为一组CPU指令后，就以文件方式被存储在硬盘中
 - ◆ 可执行文件是由CPU指令及相关数据组成的
 - ◆ 由于CPU指令及相关数据以二进制方式存储，因此可执行文件是不适合人阅读的

```
10010000001111100000000000000000 0x903E0000
10100010010010010000000000000000 0xA2490000
00100110001100010000000000000001 0x26310001
00100110010100100000000000000001 0x26520001
00010001001000000000000000000001 0x11200001
00001000000100000000000000000000 0x08100000
```

指令的机器码（2进制格式和16进制格式）

③存储



文件相关内容属于
OS技术范畴

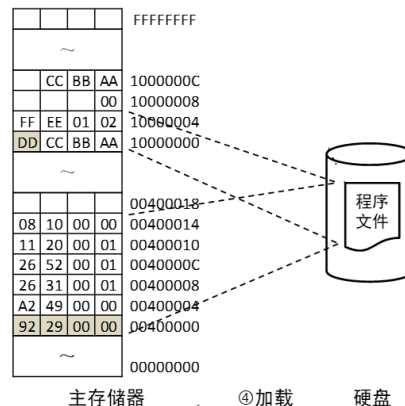
7



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

从C代码到可执行文件运行^{5/6}

- 可执行文件要被加载到主存后才能被执行。指令部分和数据部分会被分别加载到主存中的不同区域
 - ◆ 代码段：这部分主存区域存储的是CPU指令
 - ◆ 数据段：这部分主存区域存储的是数据



主存储器

④加载 硬盘

将可执行文件加载到内存
属于OS技术范畴
代码段、数据段等内容与
编译、OS都相关

8



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

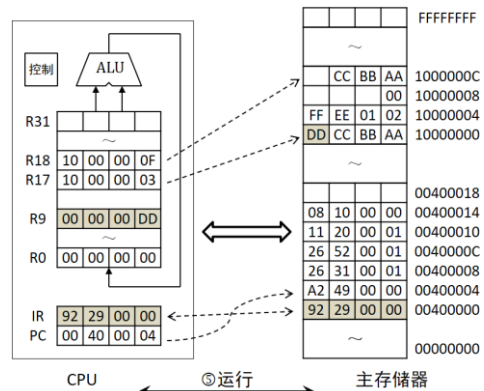
从C代码到可执行文件运行^{6/6}

□ CPU执行程序的基本原理是重复执行如下环节

- ◆ 读取指令：CPU从主存的代码段中读取一条指令到内部
- ◆ 分析指令：CPU分析指令的功能
- ◆ 执行指令：CPU控制内部的功能部件执行相应的操作

□ 读、写主存的指令

- ◆ 负责在CPU与主存间传输数据



9



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

机器语言^{1/2}

- 指令：CPU理解的“单词”
- 指令集：CPU理解的全部“单词”集合
- Q1：为什么有时不同的计算机使用相同的指令集？
 - ◆ 例如：iPhone与iPad使用相同的指令集（都是ARM）
- Q2：为什么有时不同的计算机使用不同的指令集？
 - ◆ 例如：iPhone与Macbook使用不同的指令集（前者是ARM，后者是X86）



机器语言^{2/2}

- 如果只有一种ISA
 - ◆ 可以很好的利用公共软件，如编译器、操作系统等
- 如果有多种ISA
 - ◆ 针对不同的应用可以选择更适用的ISA
 - ◆ 不同的指令集有不同的设计平衡性考虑
 - 功能、性能、存储器、功耗、复杂度。。。。
 - ◆ 会激发竞争和创新

为什么要学习汇编？

- 在更深层次理解计算机行为
 - ◆ 学习如何写更紧凑和有效的代码
 - ◆ 某些情况下，手工编码的优化水平比编译器高
- 对于资源紧张的应用，可能只适合手工汇编
 - ◆ 例如：分布式传感器应用
 - 为了降低功耗和芯片大小，甚至没有OS和编译器

RISC

- *Complex Instruction Set Computing (CISC)*
 - ◆ 指令集设计早期阶段倾向于：应用有什么操作模式，就增加对应的指令。这导致了CISC
- *Reduced Instruction Set Computing (RISC)*
 - ◆ 另一种对立的设计哲学
 - ◆ 自然界存在2-8定律。程序也类似，为什么？
- RISC的指导思想
 - ◆ 1) 加速大概率事件
 - ◆ 2) 简单意味着更容易设计、电路频率更高
 - ◆ 3) 简单功能由硬件实现；复杂功能（由大量小功能组成）交给软件处理
 - 隐含的物理背景：复杂的功能也是小概率的

13



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

RISC设计原则

- 指导思想：CPU越简单，性能越高
- 设计目标：减少指令数量，去除复杂指令（等效于降低复杂度）
- RISC的基本策略
 - ◆ 指令定长：所有指令都占用32位（1个字）
降低了从存储器中读取指令的复杂度
 - ◆ 简化指令寻址模式：以基地址+偏移为主
降低了从主存中读取操作数的复杂度
 - ◆ ISA的指令不仅数量少，而且简单
降低了指令执行的复杂度
 - ◆ 只有load与store两类指令能够访存
例如，不允许寄存器+存储器或存储器+存储器
 - ◆ 把复杂留给编译
编译器将高层语言复杂语句转换为若干简单的汇编指令

14



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

主流的ISA

- Intel 80x86
 - ◆ PC、服务器、笔记本
- ARM (Advanced RISC Machine)
 - ◆ 手机、平板
 - ◆ 出货量最大的RISC：是x86的20倍
- PowerPC
 - ◆ IBM/Motorola/Apple联盟的产物
 - ◆ 航空电子设备：飞控、机载雷达等
 - ◆ 网络设备：交换机、路由器
 - ◆ 引擎控制器

15



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

为什么选择MIPS

- 真实：工业界实际使用的CPU
 - ◆ 是设计师在实践中多次迭代、反复权衡的产物
 - ◆ 学习标准就是在学习设计师的思考方法与过程
- 简明：类别有限、结构简单、层次清晰，易于实现
 - ◆ MIPS是RISC的典型代表
- 生态：软件开发环境丰富，易于学习和实践
 - ◆ 多种模拟器、C编译等

16



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

提纲

- 机器语言概述
- 寄存器
- 指令和立即数
- 数据传输指令
- 判断指令
- 运算指令

北京航空航天大学计算机学院

计算机硬件的操作数

- C程序：变量的数量仅仅受限于内存容量
 - ◆ 程序员通常认为内存“无限大”，故声明变量时一般不考虑变量的数量
- ISA：有一组数量有限且固定的操作数，称之为寄存器
 - ◆ 寄存器被内置在CPU内部
 - ◆ 寄存器的优势：速度极快（工作速度小于1ns）
 - ◆ 寄存器的劣势：数量少



计算机中的时间单位

□ 注意：K在十进制和二进制中，并不总是1000

◆ 其他类推

时间单位	ms	μ s	ns	ps
英文全称	millisecond	microsecond	nanosecond	picosecond
中文	毫秒	微秒	纳秒	皮秒
换算	$1s=10^3ms$ $1ms=10^{-3}s$	$1s=10^6\mu s$ $1\mu s=10^{-6}s$	$1s=10^9ns$ $1ns=10^{-9}s$	$1s=10^{12}ps$ $1ps=10^{-12}s$
频率等级	KHz (Kilo, 千)	MHz (Mega, 兆)	GHz (Giga, 吉)	THz (Tera, 太)

19



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

MIPS的寄存器^{1/2}

□ MIPS寄存器数量：32

- ◆ 每个寄存器的宽度都是32位
- ◆ 寄存器没有类型（即无正负）
 - 根据指令的功能来解读寄存器值的正负
 - 即32位编码是按无符号还是符号解读，取决于指令

预留的思考题
MIPS的寄存器为
什么不是16个，
也不是64个？！

□ 寄存器数量的是**设计均衡**的体现

- ◆ 均衡的要素：性能与可用性
- ◆ 数量少：结构**简单**，速度**快**，能够存储在CPU内的数据**少**
- ◆ 数量多：结构**复杂**，速度**慢**，能够存储在CPU内的数据**多**

20



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

MIPS的寄存器^{2/2}

- 寄存器编号：0~31
- 寄存器表示：\$x（x为0~31），即\$0~\$31
- 寄存器名字
 - ◆ 程序员变量寄存器
 - \$s0-\$s7 ↔ \$16-\$23
 - ◆ 临时变量寄存器
 - \$t0-\$t7 ↔ \$08-\$15
 - \$t8-\$t9 ↔ \$24-\$25

提示
使用寄存器名字
会让代码可读性
更好

21



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

提纲

- 机器语言概述
- 寄存器
- 指令和立即数
- 数据传输指令
- 判断指令
- 运算指令

MIPS指令^{1/2}

- 指令的**一般性**语法格式：1个操作符，3个操作数

op dst, src1, src2

- ◆ op: 指令的基本功能
- ◆ dst: 保存结果的寄存器 (“destination”)
- ◆ src1: 第1个操作数 (“source 1”)
- ◆ src2: 第2个操作数 (“source 2”)

- 🔴 固定的格式：①有助于人的记忆和书写；②有助于使得硬件简单

- ◆ 硬件越简单，延迟就越小，时钟频率就越高

23



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

MIPS指令^{2/2}

- 每条指令只有1个操作
- 每行写一条指令
- 很多指令与C运算高度相关
 - ◆ 如：=, +, -, *, /, &, |
- 一行C代码会对应多条指令

24



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

MIPS指令示例^{1/2}

□ 假设：变量a, b和c分别存储在\$s1, \$s2和\$s3

◆ $a \leftrightarrow \$s1$, $b \leftrightarrow \$s2$, $c \leftrightarrow \$s3$

□ 整数加法指令

◆ C: $a = b + c$

◆ MIPS: `add $s1, $s2, $s3`

□ 整数减法指令

◆ C: $a = b - c$

◆ MIPS: `sub $s1, $s2, $s3`

25



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University


MIPS指令示例^{2/2}


□ 假设： $x \leftrightarrow \$s0$, $a \leftrightarrow \$s1$, $b \leftrightarrow \$s2$, $c \leftrightarrow \$s3$, $d \leftrightarrow \$s4$

□ C语句：`x = (a + b) - (c + d) ;`

□ MIPS汇编程序片段

执行 序 ↓	1	<code>add \$t1, \$s3, \$s4</code>	# t1 = c+d
	2	<code>add \$t2, \$s1, \$s2</code>	# t2 = a+b
	3	<code>sub \$s0, \$t2, \$t1</code>	# x = (a+b) - (c+d)


MIPS汇编程序


注释

◆ \$t1, \$t2: 临时变量寄存器



◆ 注释：提高可读性；帮助追踪寄存器/变量的分配与使用

• #: 是注释语句的开始

26



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

0号寄存器

- 由于0在程序中的频度极高，为此MIPS设置了0号寄存器

- ◆ 表示方法：\$0或\$zero
- ◆ 值恒为0：读出的值恒为0；写入的值被丢弃
 - 指令的dst为\$0：指令使用本身无错，但执行时没有实际意义

- 示例

假设：a \leftrightarrow \$s1, b \leftrightarrow \$s2, c \leftrightarrow \$s3, d \leftrightarrow \$s4

```
1  add $s3, $0, $0      # c=0
2  add $s1, $s2, $0     # a=b
```

27



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

立即数

- 指令中出现的常量数值被称为立即数
- 语法格式

op dst, src, **imm**

- ◆ 立即数替代了第2个操作数

- 示例

假设：a \leftrightarrow \$s1, b \leftrightarrow \$s2, c \leftrightarrow \$s3, d \leftrightarrow \$s4

```
1  addi $s1, $s2, 5      # a=b+5
2  addi $s3, $s3, 1      # c++
```

Q
为什么没有subi?

immediate~立即数

28



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

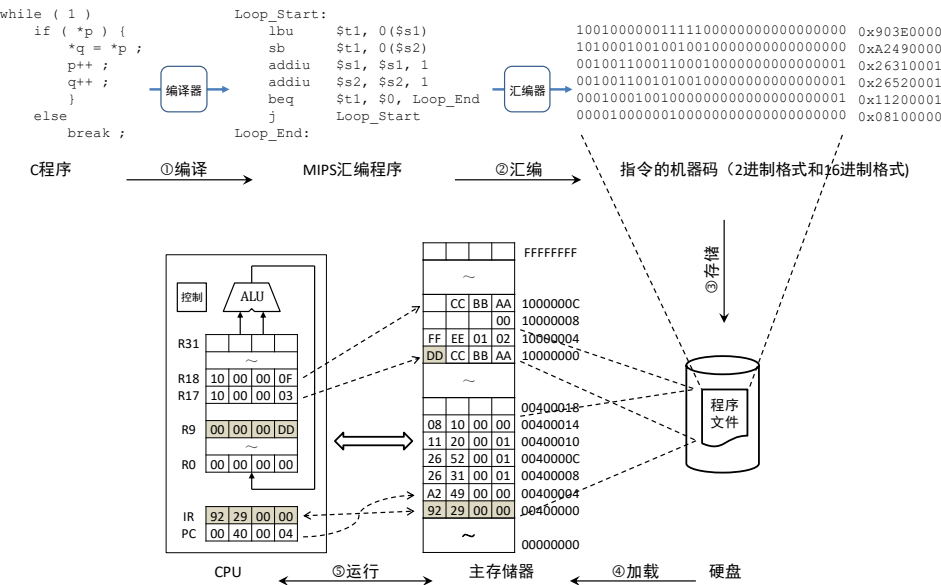
提纲

- 机器语言概述
- 寄存器
- 指令和立即数
- 数据传输指令
- 判断指令
- 运算指令

北京航空航天大学计算机学院

数据传输概览

□ 数据传输：寄存器与存储器之间的数据交换



主存单元

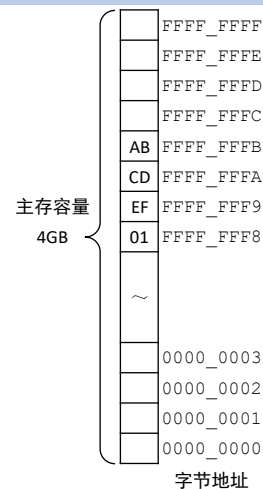
- 寄存器的局限性：有限的寄存器无法满足无限的变量需求
 - ◆ 1) 变量的个数无限：理论上，程序员可以定义任意多变量
 - ◆ 2) 变量的容量巨大：例如高维矩阵这样的大型数据结构
- 解决问题途径：主存
 - ◆ 绝大多数变量（包括数据）存储在主存中
 - ◆ 需要使用时再将其加载至寄存器中
- CPU访问主存，需要解决问题：
 - ◆ 1) CPU如何看待主存的？这就是存储视图
 - ◆ 2) CPU如何定位某个主存单元？这就是寻址方式
 - ◆ 3) CPU如何访问访问主存单元？

31


 北京航空航天大学计算机学院
 School of Computer Science and Engineering, Beihang University

主存的抽象模型^{1/3}

- 主存：可以被抽象为一个数组
 - ◆ 概念1：主存单元的颗粒度
 - 字节：是存储器最常用的存储单位
 - ◆ 概念2：地址
 - 主存单元的编号就是地址，等同于数组下标
- 对于4GB主存
 - ◆ 4GB主存共有4G个字节，即有4G个存储单元
 - ◆ 4G个存储单元对应的地址位数为32位
 - $2^{32}=4G$
 - ◆ 地址范围：0000_0000h至FFFF_FFFFh



bit~比特, byte~字节, word~字


 北京航空航天大学计算机学院
 School of Computer Science and Engineering, Beihang University

主存的抽象模型^{2/3}

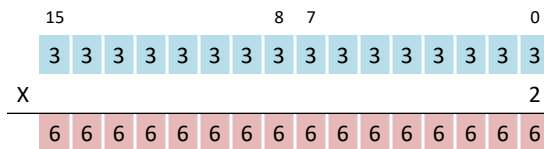
□ 概念：CPU字长

- ◆ CPU字长一般是指CPU一条指令可以计算的数据的宽度
- ◆ 例如MIPS的字长是32位，即单条MIPS指令可以计算的数据为32位

□ 注意：CPU字长只是限制了单条指令的计算能力，但是完全可以通过组合多条指令及必要的存储单元来计算更大位数的数据

□ 示例：大数乘法

- ◆ 提示：用一个数组的每个单元来存储每一位数字；然后组织循环，利用最基本的数学运算知识独立计算每位结果及其进位



用16个单元（16字节）的数组计算乘法

**循环16次，每次计算1次乘法（含进位处理）



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

主存的抽象模型^{3/3}

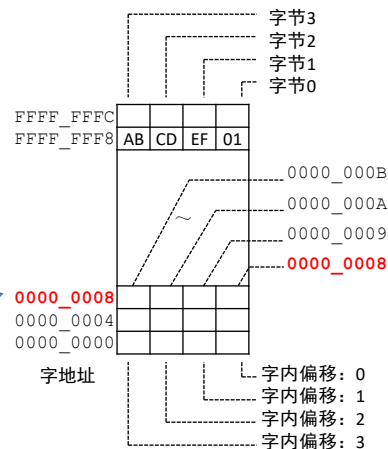
□ 为了方便从CPU的角度观察主存，经常视主存为2维数组模型

- ◆ 数组的每一行的单元数等于CPU字长
- ◆ 示例：4GB主存包含1G个字

$$4GB \div (4B/W) = 1GW$$

□ 概念4：字地址

- ◆ 字地址是指某字第0列单元的字节地址
- ◆ 示例：字2的第0列单元的地址为 0000_0008，故字2的字地址为0000_0008



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

地址的表示方式

□ 方式1：绝对地址

- 主存单元的地址直接用具体数值表示
- 示例：直接给出黄色单元的地址 $P = \text{FFFF_FFF9}$

□ 方式2：相对地址

- 主存单元的地址采用“**基地址**+**偏移**”的表示方式
- 示例1：以 $B1$ 为基地址，以 1 为偏移，即 $P = B1 + 1$
- 示例2：以 $B2$ 为基地址，以 -2 为偏移，即 $P = B2 + (-2)$

	FFFF_FFFF
	FFFF_FFFE
	FFFF_FFFD
	FFFF_FFFC
B2 →	AB FFFF_FFFB
	CD FFFF_FFFA
P →	EF FFFF_FFF9
B1 →	01 FFFF_FFF8
	~
	0000_0003
	0000_0002
	0000_0001
	0000_0000

字节地址

base address~基地址, offset~偏移



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

“基地址+偏移”的优点

□ 一致：与数据结构的访问方式高度一致

- 示例1：数组单元 $A[5]$ 的地址 = **数组首地址** + 4×5
- 示例2：以 $B.z$ 为例，可以推算出 z 与 B 的起始地址的距离
 - 为了便于分配存储器，编译器会把 x 、 y 、 z 连续存储

```
int A[100] ;

struct {
    int    x ;
    short y ;
    char   z ;
} B ;
```

□ 统一：可以用某个固定base与不同的offset计算得到任意地址

□ 灵活：不同的{base,offset}组合可以对应同一个地址，为软件编程带来很大的灵活性

- 示例：可以从数组起始向末尾遍历（base为数组第0单元地址，offset为正），也可以从数组末尾向起始遍历（base为数组最后单元地址，offset为负）



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

数据传输指令^{1/2}

□ 语法格式

op reg, off(base)

- ◆ reg: 写入或读出的寄存器
- ◆ base: 存储基地址的寄存器
 - 由于存储的是地址信息, 因此base的值被作为无符号数
- ◆ off: 以字节为单位的偏移量
 - off是立即数, 可正可负

□ 读写的存储单元的实际地址 = base + off

37



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

数据传输指令^{2/2}

□ 加载字: lw (Load Word)

- ◆ 读取地址为base+off的主存单元, 然后写入reg

□ 存储字: sw (Store Word)

- ◆ 读取reg, 然后写入地址为base+off的存储单元

□ 示例

```
int A[100] ;  
A[10] = A[3] + a ;
```

假设: $A[] \leftrightarrow \$s3$, $a \leftrightarrow \$s0$

1	lw	\$t0, 12(\$s3)	# \$t0=A[3]
2	add	\$t0, \$s0, \$t0	# \$t0=A[3]+a
3	sw	\$t0, 40(\$s3)	# A[10]=A[3]+a

注意

lw/sw读写对象是字, 因此偏移量必须是4的倍数

38



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

主存单元使用的限制

❑ MIPS不支持主存单元参与运算

- ◆ 例如以下用法均是错误用法

```
add $t0, $s1, 0($s2)
```

```
sub $t0, 0($s2), 12
```

❑ MIPS支持的运算

- ◆ 寄存器—寄存器：寄存器与寄存器运算，结果写入寄存器
- ◆ 寄存器—立即数：寄存器与立即数运算，结果写入寄存器

注意

在MIPS中，主存单元仅能与寄存器进行数据交换。
之所以这样设计，是为了便于设计流水线CPU。

39



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

lb/sb的用法^{1/3}

❑ 加载/存储字节的指令

```
lb $s0, 0($s1)
```

```
sb $s0, 1($s1)
```

提示

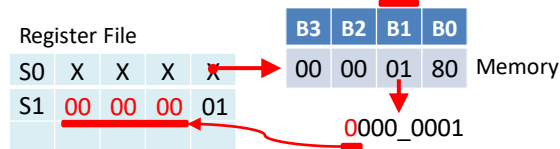
lb/sb偏移可以不是4的倍数

❑ lb：需要进行24位的符号扩展

- ◆ 读入字节的最高位被视为符号位；向高24位进行符号扩展

❑ 示例：假设*(\$s0) = 0x00000180

```
lb $s1, 1($s0) # $s1=0x00000001
```



lb/sb的用法^{2/3}

▣ 专用的加载/存储字节的指令

```
lb $s0, 0($s1)
```

```
sb $s0, 1($s1)
```

提示

lb/sb偏移可以不是4的倍数

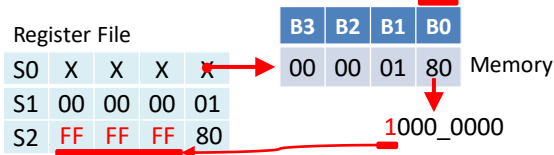
▣ lb: 需要进行24位的符号扩展

- ◆ 读入字节的最高位被视为符号位；向高24位进行符号扩展

▣ 示例：假设*(\$s0) = 0x00000180

```
lb $s1, 1($s0) # $s1=0x00000001
```

```
lb $s2, 0($s0) # $s2=0xFFFFF80
```

lb/sb的用法^{3/3}

▣ 专用的加载/存储字节的指令

```
lb $s0, 0($s1)
```

```
sb $s0, 1($s1)
```

提示

lb/sb偏移可以不是4的倍数

▣ lb: 需要进行24位的符号扩展

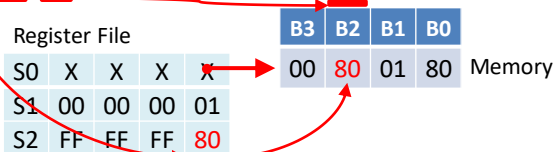
- ◆ 读入字节的最高位被视为符号位；向高24位进行符号扩展

▣ 示例：假设*(\$s0) = 0x00000180

```
lb $s1, 1($s0) # $s1=0x00000001
```

```
lb $s2, 0($s0) # $s2=0xFFFFF80
```

```
sb $s2, 2($s0) # *($s0)=0x00800180
```



加载和存储指令汇总

- 字操作：偏移必须是4的倍数
 - ◆ lw、sw
- 半字操作：偏移必须是2的倍数
 - ◆ lh、lhu
 - ◆ sh
- 字节操作
 - ◆ lb、lbu
 - ◆ sb

提示

lbu/lhu: 0扩展

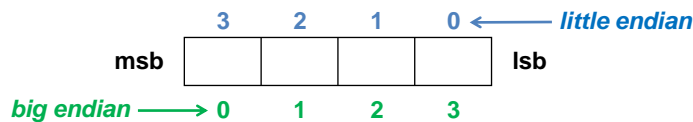
43



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

大/小印第安

- 大印第安：最高有效字节在字内的最低地址
- 小印第安：最高有效字节在字内的最高地址



- MIPS：同时支持2种类型
 - ◆ 本课程用小印第安

endianness~字节顺序

44



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

提纲

- 机器语言概述
- 寄存器
- 指令和立即数
- 数据传输指令
- 判断指令
- 运算指令

北京航空航天大学计算机学院

基本的决策机制

- C：有if-else, for, while, do-while等语句块
 - ◆ 决策机制：根据条件转移并执行相应的语句块
- MIPS：通过标号机制来实现转移
 - ◆ MIPS没有语句块的概念，只有地址的概念
 - ◆ 每条指令都对应一个word地址
 - ◆ 为了提高可读性，汇编程序使用标号来标记其后的指令的地址
 - 标号是由字符串+':'组成。例如：ForBegin:
 - ◆ 汇编语言再通过跳转机制跳转到标号处，从而实现转移

提示

C也有类似的机制，例如goto，但被认为是不好的编程风格

label~标号

46



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

决策指令

- **Branch If Equal (beq): 相等时转移**
 - ◆ `beq reg1, reg2, label`
 - ◆ 如果`reg1`的值=`reg2`的值, 则转移至`label`处执行
- **Branch If Not Equal (bne): 不等时转移**
 - ◆ `bne reg1, reg2, label`
 - ◆ 如果`reg1`的值 \neq `reg2`的值, 则转移至`label`处执行
- **Jump (j): 无条件转移**
 - ◆ `j label`
 - ◆ 无条件转移至`label`处执行
 - ◆ 对应C的goto机制

47



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

用beq构造if-else

- 与C不同之处: b类指令是条件为**TRUE**则**转移**, **FALSE**则**顺序**!

C Code:

```
if(i==j) {
    a = b /* then */
} else {
    a = -b /* else */
}
```

In English:

- 如果TRUE, 执行THEN语句块
- 如果FALSE, 执行ELSE语句块

MIPS (beq):

```
# i→$s0, j→$s1
# a→$s2, b→$s3
beq $s0, $s1, ???
??? ← 可以去除该标号
sub $s2, $0, $s3
j    end
then:
add $s2, $s3, $0
end:
```

48



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

用bne构造if-else

- 与C不同之处：beq/bne构造if-else是**条件为TRUE则转移**！

C Code:

```
if(i==j) {
    a = b /* then */
} else {
    a = -b /* else */
}
```

In English:

- 如果TRUE, 执行THEN语句块
- 如果FALSE, 执行ELSE语句块

MIPS (bne):

```
# i→$s0, j→$s1
# a→$s2, b→$s3

bne $s0,$s1,???,
???,
add $s2, $s3, $0
j    end
else:
sub $s2, $0, $s3
end:
```

49



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

switch-case

- 大多数高级程序设计语言具有switch-case语句结构
- 在MIPS汇编程序中，可以用一组b类指令来构造switch-case

```
1 switch ()
2   case 条件1
3     语句块1
4   case 条件2
5     语句块2
6   ...
7   case 条件N
8     语句块N
9
10
11
12
13
14
15
16
17
```

```
b类, 条件1, Case1
b类, 条件2, Case2
...
b类, 条件N, CaseN
j SwitchEnd

Case1:
  语句块1
  j SwitchEnd

Case2:
  语句块2
  j SwitchEnd
...
CaseN:
  语句块N
  j SwitchEnd
```

SwitchEnd :

注意1
每个case后面都
需要一条j指令

注意2
该 MIPS 汇 编 框
架 没 有 考 虑
default情况



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

更多分支指令

- 与0比较的分支指令（简称**bxxz**类指令）
 - ◆ blez（小于等于0转移）
 - ◆ bgtz（大于0转移）
 - ◆ bltz（小于0转移）
 - ◆ bgez（大于等于0转移）
- **bxxz**类指令格式均为：**bxxz rs, label**
 - ◆ 由于**bxxz**指令固定与\$0比较，因此指令中无需再描述\$0了



循环

- C语言有3种循环：for, while, do...while
 - ◆ 3种语句是等价的，即任意一种循环都可以改写为其他两种循环
- MIPS只需要一种决策机制即可
 - ◆ 核心：根据条件转移



实战：从C到MIPS

▣ 实例：字符串赋值

▣ C代码

```
/* Copy string from p to q */
char *p, *q;
while ((*q++ = *p++) != '\0') ;
```

▣ 代码结构有什么特征

- ◆ 单一的while循环
- ◆ 退出循环是一个相等测试

53



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

实战：从C到MIPS

▣ 实例：字符串赋值

▣ C代码

```
/* Copy string from p to q */
char *p, *q;
while ((*q++ = *p++) != '\0') ;
```

▣ 代码结构有什么特征？

- ◆ 单一的while循环
- ◆ 退出循环是一个相等测试

Q: 2种写法的区别在哪里？

```
while ( *p )
    *q++ = *p++ ;
```

实战：从C到MIPS

▣ STEP1: 构造循环的框架

```
# copy String p to q
# p→$s0, q→$s1 (pointers)
Loop:                                # $t0 = *p
                                     # *q = $t0
                                     # p = p + 1
                                     # q = q + 1
                                     # if *p==0, go to Exit
                                     # go to Loop
        j Loop
Exit:
```

55



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

实战：从C到MIPS

▣ STEP2: 构造循环主体

```
# copy String p to q
# p→$s0, q→$s1 (pointers)
Loop: lb    $t0,0($s0)    # $t0 = *p
      sb    $t0,0($s1)    # *q = $t0
      addi  $s0,$s0,1      # p++
      addi  $s1,$s1,1      # q++
      beq   $t0,$0,Exit    # if $t0==0, go to Exit
      j     Loop          # go to Loop
Exit:
```

Q
1个字符需要6条指令。能否优化？

56

实战：从C到MIPS

- 优化代码（减少了1条指令）

copy String p to q

p→\$s0, q→\$s1 (pointers)

```
Loop: lb    $t0, 0($s0)    # $t0 = *p
      sb    $t0, 0($s1)    # *q = $t0
      addi  $s0, $s0, 1    # p = p + 1
      addi  $s1, $s1, 1    # q = q + 1
      bne   $t0, $0, Loop  # if *p!=0, go to Loop
```

57



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

提纲

- 机器语言概述
- 寄存器
- 指令和立即数
- 数据传输指令
- 判断指令
- 运算指令



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

算术运算指令概述

- 计算机最重要的功能就是完成加、减、乘及除等算术运算
- 几乎所有的高级程序设计都支持这些算术运算
- MIPS为此设置相应的算术运算指令
- 除从运算功能角度分类，还可从操作数角度分类
 - ◆ 寄存器—寄存器型（R-R）：参与运算的2个操作数都是寄存器，结果写入寄存器
 - ◆ 寄存器—立即数型（R-I）：参与运算的2个操作数一个是寄存器，另一个是立即数，结果写入寄存器



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

加法和减法

- 加法有4条指令：add, addu, addi, addiu
- 减法有2条指令：sub, subu

指令	功能	格式	描述	示例
add	加法 (检测溢出)	add rd, rs, rt	$R[rd] \leftarrow R[rs] + R[rt]$	add \$s1, \$s2, \$s3
addu	加法 (不检测溢出)	add rd, rs, rt	$R[rd] \leftarrow R[rs] + R[rt]$	addu \$s1, \$s2, \$s3
addi	立即数加 (检测溢出)	addi rt, rs, imm16	$R[rt] \leftarrow R[rs] + \text{sign_ext}(\text{imm16})$	addi \$s1, \$s2, -3
addiu	立即数加 (不检测溢出)	addiu rt, rs, imm16	$R[rt] \leftarrow R[rs] + \text{sign_ext}(\text{imm16})$	addiu \$s1, \$s2, 3
sub	减法 (检测溢出)	sub rd, rs, rt	$R[rd] \leftarrow R[rs] - R[rt]$	sub \$s1, \$s2, \$s3
subu	减法 (不检测溢出)	sub rd, rs, rt	$R[rd] \leftarrow R[rs] - R[rt]$	subu \$s1, \$s2, \$s3

问题
为什么没有subi和subiu？



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

算术溢出^{1/2}


- 复习：当计算结果的位数超出计算机硬件的实际能保存的位数，即为**溢出**
 - ◆ 换言之，即没有足够的位数保存运算结果
- MIPS会检测溢出（并且当溢出发生时**产生错误**）
 - ◆ 有unsigned关键字的算术类指令忽略溢出

检测溢出	不检测溢出
add dst,src1,src2	addu dst,src1,src2
addi dst,src1,src2	addiu dst,src1,src2
sub dst,src1,src2	subu dst,src1,src2

61

算术溢出^{1/2}

□ 示例


复习：这是最小的负数！

```

# $s0=0x80000000, $s1=0x1
add    $t0,$s0,$s0 # 溢出（出错）
addu   $t1,$s0,$s0 # $t1=0
addi   $t2,$s0,-1  # 溢出（出错）
addiu  $t3,$s0,-1  # $t3=0x7FFFFFFF
sub     $t4,$s0,$s1 # 溢出（出错）
subu   $t5,$s0,$s1 # $t5=0x7FFFFFFF
  
```

62



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

乘除法指令

- 乘除法指令计算结果：不是直接写入32个通用寄存器，而是保存在2个特殊寄存器HI与LO
- 用2条专用指令读写HI/LO
 - “move from HI” (mfhi dst)
 - “move from LO” (mflo dst)
- **Multiplication** (mult)
 - ◆ `mult src1,src2`
 - ◆ `src1*src2`: LO保存结果的低32位, HI保存结果的高32位
- **Division** (div)
 - ◆ `div src1,src2`
 - ◆ `src1/src2`: LO保存商, HI保存余数

quotient~商; remainder~余数

63



北京航空航天大学计算机学院
School of Computer Science and Engineering, Beihang University

乘除法指令

- 示例：用div求模
- ```
$s2 = $s0 mod $s1
mod:
div $s0,$s1 # LO = $s0/$s1
mfhi $s2 # HI = $s0 mod $s1
```

64



北京航空航天大学计算机学院  
School of Computer Science and Engineering, Beihang University



## 位运算指令

□ 假设：  $a \rightarrow \$s1, b \rightarrow \$s2, c \rightarrow \$s3$

| Instruction            | C                       | MIPS                              |
|------------------------|-------------------------|-----------------------------------|
| And                    | $a = b \ \& \ c;$       | <code>and \$s1, \$s2, \$s3</code> |
| And Immediate          | $a = b \ \& \ 0x1;$     | <code>andi \$s1, \$s2, 0x1</code> |
| Or                     | $a = b \   \ c;$        | <code>or \$s1, \$s2, \$s3</code>  |
| Or Immediate           | $a = b \   \ 0x5;$      | <code>ori \$s1, \$s2, 0x5</code>  |
| Not Or                 | $a = \sim(b \   \ c);$  | <code>nor \$s1, \$s2, \$s3</code> |
| Exclusive Or           | $a = b \ \wedge \ c;$   | <code>xor \$s1, \$s2, \$s3</code> |
| Exclusive Or Immediate | $a = b \ \wedge \ 0xF;$ | <code>xori \$s1, \$s2, 0xF</code> |

65

## 移位指令

□ C语言有移位操作，MIPS也定义了多条移位指令

□ 移位指令可以从3个维度来分析

- ◆ 方向：左移还是向右移
- ◆ 性质：逻辑移位还是算术移位
  - 对于向左移位来说，低位永远是补0
  - 只有向右移位，才存在高位是补0还是符号位的选择问题。如果补0，那就是逻辑移位，如果是补符号位，则为算术移位。
- ◆ 移位量：对于32位寄存器，移动位数的合理最大取值为31，即0x1F
  - 如何在指令中表示这个移位量呢？
  - 方式1：由一个5位的立即数来表示移位量
  - 方式2：用某寄存器的值来表示移位量。如果用寄存器来表示移位量，则只有该寄存器的最低5位被CPU识别为移位量，而高27位无论取何值均无意义。

Q

根据上述3个维度，应该定义几条指令？

66



北京航空航天大学计算机学院  
School of Computer Science and Engineering, Beihang University

## 移位指令

### □ MIPS共6条移位指令

- ◆ 如果使用立即数：只有0~31有效
- ◆ 如果使用寄存器：寄存器的低5位有效（按5位无符号数对待）

| 指令   | 功能     | 示例                    |
|------|--------|-----------------------|
| sll  | 逻辑左移   | sll \$t0, \$s0, 16    |
| srl  | 逻辑右移   | srl \$t0, \$s0, 16    |
| sra  | 算术右移   | sra \$t0, \$s0, 16    |
| sllv | 逻辑可变左移 | sllv \$t0, \$s0, \$s1 |
| srlv | 逻辑可变右移 | srlv \$t0, \$s0, \$s1 |
| srav | 算术可变右移 | srav \$t0, \$s0, \$s1 |

67


 北京航空航天大学计算机学院  
 School of Computer Science and Engineering, Beihang University

## 移位指令

### □ 示例

```

addi $t0,$0 ,-256 # $t0=0xFFFFFFFF00
sll $s0,$t0,3 # $s0=0xFFFFF800
srl $s1,$t0,8 # $s1=0x00FFFFFF
sra $s2,$t0,8 # $s2=0xFFFFFFFF

addi $t1,$0 ,-22 # $t1=0xFFFFFEEA
 # low 5: 0b01010
sllv $s3,$t0,$t1 # $s3=0xFFFC0000
same as sll $s3,$t0,10

```

68


 北京航空航天大学计算机学院  
 School of Computer Science and Engineering, Beihang University